University of Pittsburgh

Matrix Solutions for Linear Systems

Week 6

MEMS 1140—Introduction to Programming in Mechanical Engineering

Learning Objectives (L.O.)

At the end of this lecture, you should understand/be able to:

- ❑ What a linear system is;
- ❑ Format a linear system into matrix-vector form;
- ❑ Solve a linear system in matrix-vector form.
- ❑ The advantages of backslash notation for linear systems.

Table of Contents (ToC)

1 – What is a Linear System

A system of equations is defined by two or more equations with two or more unknown variables.

A linear system is linear with respect to the unknown variables.

For example:

$$\begin{cases} 2x + y = 10 \\ x + y = 6 \end{cases}$$

This system contains two equations and two variables.

1 – Constant “Planes”

Every equation in linear systems represents a constant “plane” in the space of the unknown variables.

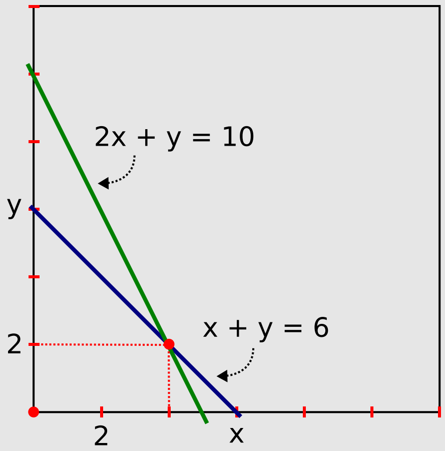
In our example, x and y are the two unknown variables. So both equations represent a “plane” in 2D — just a line.

The intersection of those “planes” is the solution to the system, if it exists.

1 – Plotting 2D Lines

Here we see the graphs of both lines in the system.

The intersection point at $(4, 2)$ is the solution to the linear system.



1 – 3D Linear Systems

Now consider the following 3D system of equations:

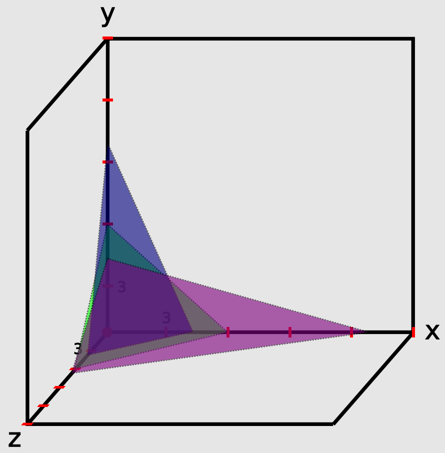
$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases}$$

Here, each equation represents a planar surface in *xyz*-space.

1 – Planar Representation

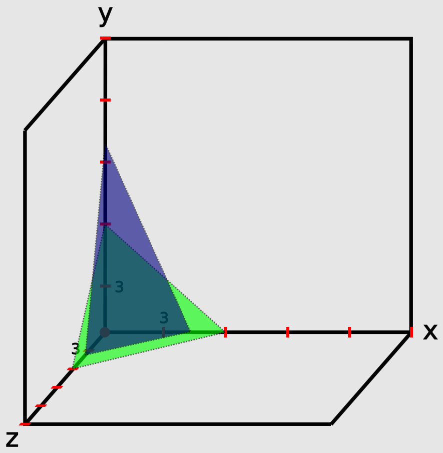
Here we see the graphs of all three planes in the system.

Note: this drawing is not perfectly to scale.



1 – Planar Representation

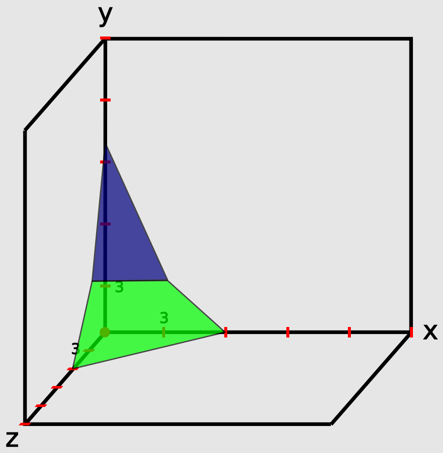
If we isolate the blue and green planes:



1 – Planar Representation

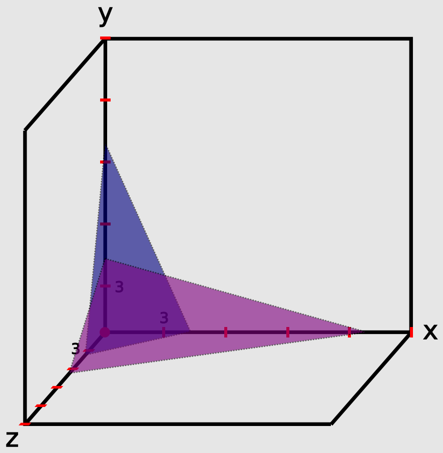
If we isolate the blue and green planes:

Highlight the line at which they intersect with each other.



1 – Planar Representation

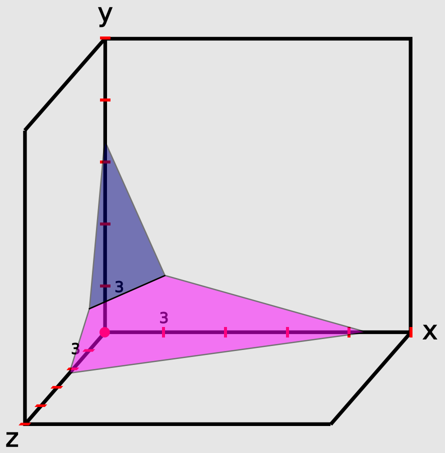
Then isolate the blue and magenta planes:



1 – Planar Representation

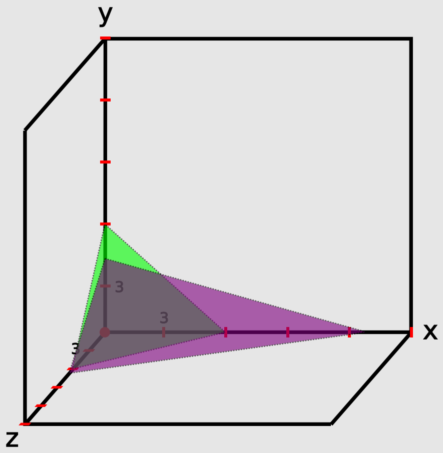
Then isolate the blue and magenta planes:

And highlight the line at which they intersect with each other.



1 – Planar Representation

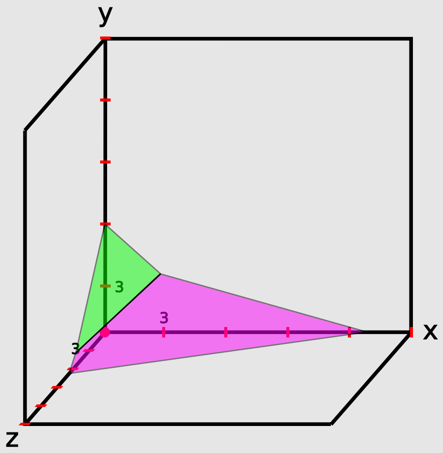
And finally isolate the green and magenta planes:



1 – Planar Representation

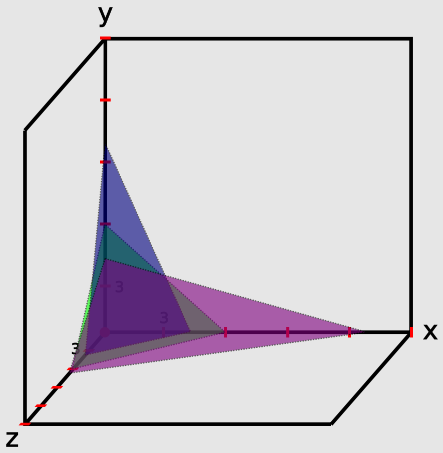
And finally isolate the green and magenta planes:

Again, highlight the line at which they intersect with each other.



1 – Planar Representation

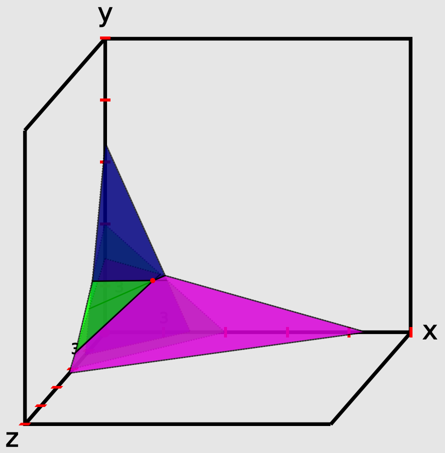
Now we can return to the view with all three planes.



1 – Planar Representation

Now we can return to the view with all three planes.

Overlaying the intersection lines, the point at which they intersect is the solution of the system.



1 – Understanding Linear Systems

This apparent tangent gives us a physical representation for linear systems of equations.

And we also now have motivation for what it means to “solve” the system.

If a solution exists, the value of each variable represents the coordinate in N-dimensional space of the intersection between each “plane.”

2 – Matrix-Vector Form

This lecture will focus on rewriting the equations in the form of matrix-vector multiplication to solve the system.

For our 3D example:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases} \rightarrow \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Matrix-Vector Form

This lecture will focus on rewriting the equations in the form of matrix-vector multiplication to solve the system.

For our 3D example:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases} \rightarrow \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Matrix-Vector Form

This lecture will focus on rewriting the equations in the form of matrix-vector multiplication to solve the system.

For our 3D example:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases} \rightarrow \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Matrix-Vector Form

This lecture will focus on rewriting the equations in the form of matrix-vector multiplication to solve the system.

For our 3D example:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases} \rightarrow \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Matrix-Vector Form

This lecture will focus on rewriting the equations in the form of matrix-vector multiplication to solve the system.

For our 3D example:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases} \rightarrow \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Matrix-Vector Form

This lecture will focus on rewriting the equations in the form of matrix-vector multiplication to solve the system.

For our 3D example:

$$\begin{cases} 2x + y + 3z = 10 \\ x + y + z = 6 \\ x + 3y + 2z = 13 \end{cases} \rightarrow \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Verification

This is equivalent to the original system of equations; follow matrix-vector multiplication for each row of the matrix.

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{cases} 2x + 1y + 3z \\ 1x + 1y + 1z \\ 1x + 3y + 2z \end{cases} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Verification

This is equivalent to the original system of equations; follow matrix-vector multiplication for each row of the matrix.

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{cases} 2x + 1y + 3z \\ 1x + 1y + 1z = \\ 1x + 3y + 2z \end{cases} \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Verification

This is equivalent to the original system of equations; follow matrix-vector multiplication for each row of the matrix.

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{cases} 2x + 1y + 3z \\ 1x + 1y + 1z \\ 1x + 3y + 2z = \end{cases} \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

2 – Verification

This is equivalent to the original system of equations; follow matrix-vector multiplication for each row of the matrix.

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{cases} 2x + 1y + 3z \\ 1x + 1y + 1z \\ 1x + 3y + 2z \end{cases} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

Matrix-vector form is *the same* as writing it as a linear system.

3 – Solving an Algebraic Equation

Let's take a slight detour to motivate the next step.

Consider how to solve for x in the following equation:

$$Ax = b$$

By algebra, we divide by A:

$$x = \frac{b}{A}$$

3 – Linear Algebra Analog

Returning to our example, each part of the matrix-vector form is labeled as follows:

$$\underbrace{\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix}}_{\mathbb{A}} \underbrace{\begin{bmatrix} x \\ y \\ z \end{bmatrix}}_{\vec{x}} = \underbrace{\begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}}_{\vec{b}} \quad \rightarrow \quad \mathbb{A}\vec{x} = \vec{b}$$

Just like with simple algebra, this system can be solved by moving $\mathbb{A}$ over to the right-hand side.

3 – Solving a Linear System

But unlike before, we don't simply divide by $\mathbb{A}$. Matrices *cannot* be divided!

Instead, we ***invert*** the matrix:

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

3 – Inverting the Matrix

The hardest part of this process is calculating the inverse:

$$\begin{bmatrix} 2 & 1 & 3 \\ 1 & 1 & 1 \\ 1 & 3 & 2 \end{bmatrix}^{-1} = \underbrace{\frac{1}{3}\begin{bmatrix} -1 & 7 & -2 \\ -1 & 1 & 1 \\ 2 & -5 & 1 \end{bmatrix}}_{\mathbb{A}^{-1}}$$

I used MATLAB for this matrix, but there are methods for inverting matrices by hand. Feel free to read more!

3 – Solving By Hand

With the inverse matrix calculated, let’s solve the system:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{3} \begin{bmatrix} -1 & 7 & -2 \\ -1 & 1 & 1 \\ 2 & -5 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

3 – Solving By Hand

First, solve for *x*:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{3}\begin{bmatrix} -1 & 7 & -2 \\ -1 & 1 & 1 \\ 2 & -5 & 1 \end{bmatrix}\begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

$$x = \frac{(-1)(10) + (7)(6) + (-2)(13)}{3} = \frac{-10 + 42 + -26}{3} = \frac{6}{3} = 2$$

3 – Solving By Hand

Then, solve for y:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{3} \begin{bmatrix} -1 & 7 & -2 \\ -1 & 1 & 1 \\ 2 & -5 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

$$y = \frac{(-1)(10) + (1)(6) + (1)(13)}{3} = \frac{-10 + 6 + 13}{3} = \frac{9}{3} = 3$$

3 – Solving By Hand

Finally, solve for z:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \frac{1}{3} \begin{bmatrix} -1 & 7 & -2 \\ -1 & 1 & 1 \\ 2 & -5 & 1 \end{bmatrix} \begin{bmatrix} 10 \\ 6 \\ 13 \end{bmatrix}$$

$$z = \frac{(2)(10) + (-5)(6) + (1)(13)}{3} = \frac{20 + -30 + 13}{3} = \frac{3}{3} = 1$$

3 – System Solution

And with that, the solution is given in the following vector:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$

There are better methods for solving a linear system. Cramer's rule, Gaussian elimination, and LU decomposition are some.

3 – Not Going into detail

We won't cover these methods in detail; that is a discussion for a linear algebra course.

It is valuable to understand them nonetheless.

Some advanced coursework may even expect you to implement them yourself.

We get to use MATLAB to solve the system for us!

3 – Solving in MATLAB

First, define the coefficient matrix:

```
>> coefficient_matrix = [2, 1, 3; ...
                         1, 1, 1; ...
                         1, 3, 2];      % coeff. matrix should be square
.
.
```

Command Window

3 – Solving in MATLAB

Then, define the right-hand-side vector:

```
>> coefficient_matrix = [2, 1, 3; ...
                         1, 1, 1; ...
                         1, 3, 2];
>> rhs_vector = [10; 6; 13];          % this should be a column vector!
.
```

Command Window

3 – Solving in MATLAB

Finally, solve the system:

```
>> coefficient_matrix = [2, 1, 3; ...
                         1, 1, 1; ...
                         1, 3, 2];
>> rhs_vector = [10; 6; 13];
>> solution = inv(coefficient_matrix) * rhs_vector  % calculate solution
solution =
    2.0000
    3.0000
    1.0000
```

Command Window

3 – Inverse in MATLAB

The **`inv`** command was used to evaluate the inverse of a matrix in this example.

Raising the matrix to the power **`A^(-1)`** is equivalent to using the **`inv`** command in MATLAB.

But the ***recommended*** approach to solving linear systems is to use the backslash operator instead: **`A\b`**

3 – Using the Backslash

Backslash notation **`A\b`** automatically solves a linear system:

$$\mathbb{A}\vec{x} = \vec{b}$$

This is recommended over **`inv(A)*b`** command or **`A^(-1)*b`** because it is significantly faster.

This Mathworks reference details the backslash operator.

We'll run an experiment to highlight the speed difference.

4 – Setting up the Experiment

First declare **`N`** as the number of unknowns in the system:

```
N = 1e3; .                     .                            % 1000 elements
.


.
.
.


.
.
.
```

4 – Setting up the Experiment

Define a random **NxN** coefficient matrix:

```
N = 1e3; A = rand(N, N);  .                  % 1000x1000 coeff. matrix
.

.
.
.

.
.
.
```

4 – Setting up the Experiment

Define the **`Nx1`** vector of “unknowns” to solve for $\vec{b}$ later:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);        % 1000x1 "unknowns"
.


.
.
.


.
.
.
```

4 – Setting up the Experiment

Solve for $\vec{b}$ to use in the experiment (also **`Nx1`**):

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;                              % solve for the rhs vector

.
.
.

.
.
.
```

4 – Setting up the Experiment

Start a timer for using the **`inv`** command:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;

inverse_timer = tic;                        % start a timer
.
.


.
.
.
```

4 – Setting up the Experiment

Solve the linear system using the **`inv`** command **`inv(A)*b`**:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;

inverse_timer = tic;
x_inv = inv(A) * b;                                % solve the system (inv)
.

.
.
.
```

4 – Setting up the Experiment

End the timer for using the **`inv`** command:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;

inverse_timer = tic;
x_inv = inv(A) * b;
inverse_time_elapsed = toc(inverse_timer)     % end the timer

.
.
.
```

4 – Setting up the Experiment

Start a timer for using the backslash notation **A\b**:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;

inverse_timer = tic;
x_inv = inv(A) * b;
inverse_time_elapsed = toc(inverse_timer)

backslash_timer = tic;                          % start a timer
.
.
```

4 – Setting up the Experiment

Solve the system using the backslash notation **`A\b`**:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;

inverse_timer = tic;
x_inv = inv(A) * b;
inverse_time_elapsed = toc(inverse_timer)

backslash_timer = tic;
x_backslash = A \ b;                           % solve the system (\)
.
```

4 – Setting up the Experiment

End the timer for using backslash notation:

```
N = 1e3; A = rand(N, N); x = rand(N, 1);
b = A * x;

inverse_timer = tic;
x_inv = inv(A) * b;
inverse_time_elapsed = toc(inverse_timer)

backslash_timer = tic;
x_backslash = A \ b;
backslash_time_elapsed = toc(backslash_timer) % end the timer
```

4 – Experiment Results

The values of the two timers — in seconds — are as follows:

```
inverse_time_elapsed =
     0.0519
.
.
```

Command Window Output

4 – Experiment Results

The values of the two timers — in seconds — are as follows:

```
inverse_time_elapsed =
     0.0519
backslash_time_elapsed =
     0.0207
```

Command Window Output

The ratio between these is calculated as follows:

```
>> inverse_time_elapsed / backslash_time_elapsed
.
```

Command Window

4 – Experiment Results

The values of the two timers — in seconds — are as follows:

```
inverse_time_elapsed =
     0.0519
backslash_time_elapsed =
     0.0207
```

Command Window Output

The ratio between these is calculated as follows:

```
>> inverse_time_elapsed / backslash_time_elapsed
ans = 2.5050
```

Command Window

4 – Experiment Results

The values of the two timers — in seconds — are as follows:

```
inverse_time_elapsed =
      0.0519
backslash_time_elapsed =
      0.0207
```

Command Window Output

The ratio between these is calculated as follows:

```
>> inverse_time_elapsed / backslash_time_elapsed
ans = 2.5050    % using A \ b is 2.5 times faster than inv(A)*b
```

Command Window

4 – Experiment Takeaways

When looking to evaluate just the inverse of a matrix, the `inv(A)` command or the exponent `A^(-1)` are both sufficient.

But for finding the solution to a linear system, it is faster (and more reliable) to use backslash notation.

But for small systems, time may be negligible either way:

```
x = inv(A) * b;
```

```
x = A \ b;
```

4 – Other Advantages

Backslash notation can also solve overconstrained systems, where $(N_{\text{eq}} > N_{\text{var}})$.

In contrast, **`inv(A)`** will error if **`A`** is not square, even if there might be a solution for the overconstrained case.

You may not always be solving properly constrained systems, so it is good practice to always use backslash notation.

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

This lecture covered:

✓ What a linear system is

A set of N equations with N total unknown variables. The solution to the system is the intersection point in N-D space between each “plane” in the system.

5 – Summary

✓ How to format a linear system into matrix-vector form

With each equation arranged with the unknowns in the same order, the coefficient matrix is composed of the coefficients for each variable. The variable column vector contains all unknowns. And the right-hand-side vector contains the constants.

✓ How to solve a linear system

We use MATLAB's built-in **`inv(A)*b`** or backslash notation **`A\b`** to find the solution to the linear system.

5 – Summary

✓ L.O.1
✓ L.O.2
✓ L.O.3
✓ L.O.4

✓ The advantages of backslash notation

It is faster to use backslash notation than the **`inv`** command. It can also solve overconstrained systems and it is more reliable with systems with high condition numbers, as you will find in the documentation.